TELE-CODER
A SYSTEM FOR CODING AND DECODING PROGRAMMING LANGUAGES FOR USE WITH A PUSH BUTTON TELEPHONE

JOHN NORMAN McKAY, JR.

LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIF. 93940

United States
Naval Postgraduate School

LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIF. 93940

THESIS

Tele-Coder
A System for Coding and Decoding Programming Languages for Use With a Push Button Telephone

by

John Norman McKay, Jr.

Thesis Advisor: R. C. Bolles

September 1971

Approved for public release; distribution unlimited.

LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIF. 93940

Tele-Coder

A System for Coding and Decoding Programming Languages for Use With a Push Button Telephone

by

John Norman McKay, Jr.
Lieutenant Commander, United States Navy
B.S., Austin Peay State College, 1959

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 1971

ABSTRACT

The push button telephone has been employed in many systems as a computer terminal with a very restricted capability and in at least one instance as a computer terminal for a general purpose programming language. This thesis discusses the input, output, coding, and decoding problems when considering a general purpose programming language for use with a push button telephone. Included is a general discussion of a system that would use a syntactic analysis and the context of a programming language to produce a code for use with a push button telephone. The output from this analysis would be used to build a table-driven translator to decode the language. An example of such a system is also included.

TABLE OF CONTENTS

I. INTRODUCTION ---------- 5

A. INPUT AND CODING ---------- 6

B. OUTPUT ---------- 9

II. GENERAL DISCUSSION ---------- 12

III. TELE-CODER ---------- 16

A. THE ANALYZER ---------- 16

B. THE TRANSLATOR ---------- 20

C. TESTING ---------- 22

IV. CONCLUSION ---------- 24

APPENDIX A INPUT TO ANALYZER ---------- 25

APPENDIX B INPUT TO TRANSLATOR ---------- 28

OUTPUT FROM ANALYZER ---------- 30

ANALYZER PROGRAM LISTING ---------- 42

TRANSLATOR PROGRAM LISTING ---------- 50

BIBLIOGRAPHY ---------- 59

INITIAL DISTRIBUTION LIST ---------- 60

FORM DD 1473 ---------- 61

ACKNOWLEDGEMENT

The author wishes to express his appreciation for the assistance and encouragement given him by Instructor Robert C. Bolles of the U. S. Naval Postgraduate School, whose direction and prompting in the planning stages were responsible for the depth and success of the total research effort. The author would also like to thank his wonderful wife, Charlene, for the hours she spent typing the smooth draft.

1. INTRODUCTION

The computer terminal of recent years has been some form of the common teletypwriter. This sort of device is economically practical where it serves a large number of users, but for the individual its cost precludes widespread use. The cost of such devices may be reduced significantly in the future, but it is hard to visualize a terminal that will be economically acceptable to the general user. The expense of such a device will prevent private use for all but a select few. An obvious answer for the general computer user is to employ the telephone in a secondary role as a computer terminal.

Access to the computer through the telephone will surely not appeal to all users. The obvious use would be in providing "desk calculator" type services. The company or university with an existing computer facility could provide such services at a nominal additional cost. Computer services could be sold to the general public on a timed basis just as long distance telephone service is sold. Martin [Ref. 1] predicts that this public use of the computer will have a profound effect on the computer industry.

The traditional telephone with a circular dial does not lend itself to use as an input device. The additional time and inconvenience required to dial a nine as compared to a one, would disgruntle the most patient user. The telephone companies of this nation have provided a device far superior in speed and acceptability. The "push button" or "Touch-Tone" telephone is being introduced to more areas

each year [Ref. 2]. These push button telephones significantly enhance the acceptability of the telephone as a computer terminal.

The telephone has already been used as a computer terminal on many remote access computing systems [Ref. 3] . Information retrieval systems that provide such services as stock market quotations and inventory availability have appeared in recent years. Experiments have also been conducted with systems that offer a simple computing capability similiar to that of a "desk calculator" [Ref. 4]. These systems in general require a limited input and response capability and have a well defined input and output format. The coding and decoding of a general purpose programming language for input via a push button telephone presents a more formidable problem. This research was directed at problems of this nature.

A. INPUT AND CODING

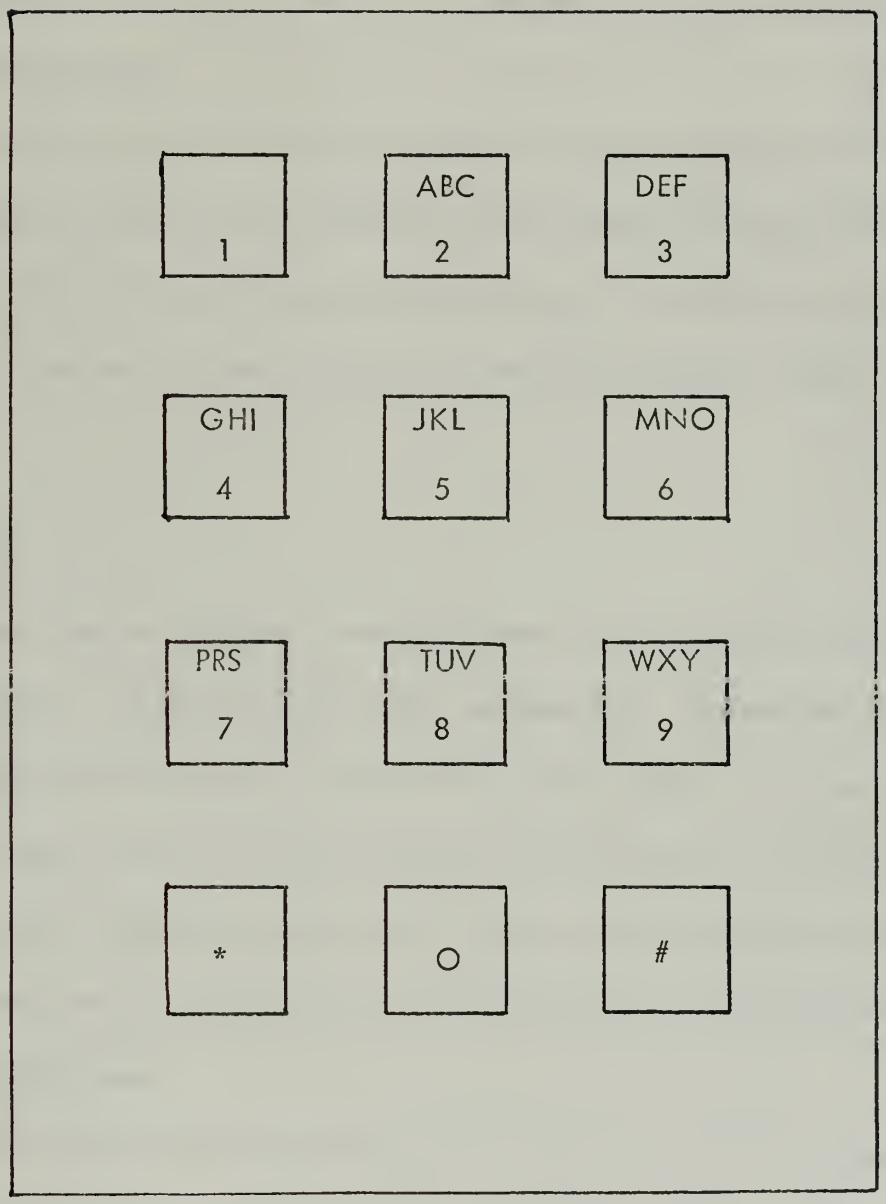
The realization of a natural language processor will some day make the telephone a very attractive input device. If the computer could "understand" spoken text, then the primary input capability of the telephone would make it desirable as a computer terminal. Communication, however, is limited by the intelligence of the least participant, and so, until the computer can converse reasonably well, the secondary input capability of the telephone will provide the only possible alternative. The problem of input then becomes one of coding a set of approximately fifty characters into a code alphabet of twelve characters. The fifty characters allow for the twenty-six letters of the alphabet, ten digits and sixteen special characters, such as ;, *, -, etc.

The push button telephone offered by the common carriers in the United States is a twelve button device. The general appearance of a push button keyboard is

given in Figure 1. The original push button telephones were ten button devices, but were designed on a four by three array with positions four-one and four-three blank. The two additional buttons have been added to the keyboard as a "no cost option" [Ref. 4]. These two additional buttons greatly simplify the input coding schemes.

The alphabetic synonyms assigned to each push button of the telephone add nothing to the information transmission of each key, but then do serve a useful purpose. Originally these letters served to aid in remembering a central office code, making the memorization of telephone numbers easier. They could serve the same purpose when considering a programming language code. The user would not be required to remember that code "77468" represented the word "PRINT". The relative location of the letters on the keys could also assist in the coding scheme of a full alphabetic input. For example, the code "12" could represent the first letter on the second key. The letters "Q" and "Z" were omitted from the telephone dial originally because there were no central office names that began with these letters. They would obviously be required for some programming languages. For purposes of this study, they were considered to be added to key "O". This leaves the two keys "*" and "#" for special functions alone, and this appears to be their best logical use.

The code word lengths are an important part of any coding scheme. Considering an information source of approximately fifty characters and a code alphabet of twelve characters, code word lengths equal to or greater than two will satisfy the conditions of the Kraft inequality (Kraft 1949) [Ref. 5]. Therefore there



PUSH BUTTON TELEPHONE KEYBOARD

FIGURE 1

exists a unique coding scheme, using two code characters per source character.[1] The use of the context of a programming language will however allow a significant reduction in the average length of a code word, and this reduction should be a primary objective.

Physically, the input would be accepted by an audio response unit at the computer facility, which would relay the digital information to the central processor [Ref. 6]. The actual processing would then be handled by an incremental compiler or translator that would respond to special codes from the user. (See Figure 2)

B. OUTPUT

The audio response equipment mentioned above is also capable of generating verbal messages. The audio output section operates from a prerecorded vocabulary, which may be changed or expanded to a limited degree. A coded message from the computer causes the proper selection of words from the vocabulary, and the response unit sends the verbal output to the conventional telephone receiver. The telephone can thus provide a limited output capability at no additional cost to the individual user.

Output should also be handled through the normal installation facilities. The user would select this type of output and his job would be placed in the normal job stream. To take the best advantage of the system, the initial translation should still be on a line-by-line basis with audio response. This would allow error detection and correction before placing the job in the normal job stream.

---

[1] It can be shown that many such schemes exist.

C. OBJECTIVES

Cornell University has implemented a remote access computing system that employs push button telephones as terminals [Ref. 4]. The system is based on CUPL, which is a batch processing system used for introductory instruction at the university. To transfer the system to another location would require the introduction of the programming language CUPL or the adaptation of the concepts of the system to a new language. This would be time consuming and costly. Instead of such an adaptation, it was felt that a generalized system could be designed that would automatically build a decoder for a push button telephone system. Figure 2 is a flow chart showing where such a system would fit into a computer processing network. The main objective of this research project was to design and test such a system.

In design of the system, it was decided that as large a class of languages as possible should be acceptable for analysis. This dictated that generality must be preserved wherever possible. The system was also to produce an average code word length as small as possible in order to minimize the button pushing required of the user. Context was to be used in pursuing this objective, and a two-key code for each input character was to be considered only as a last resort.

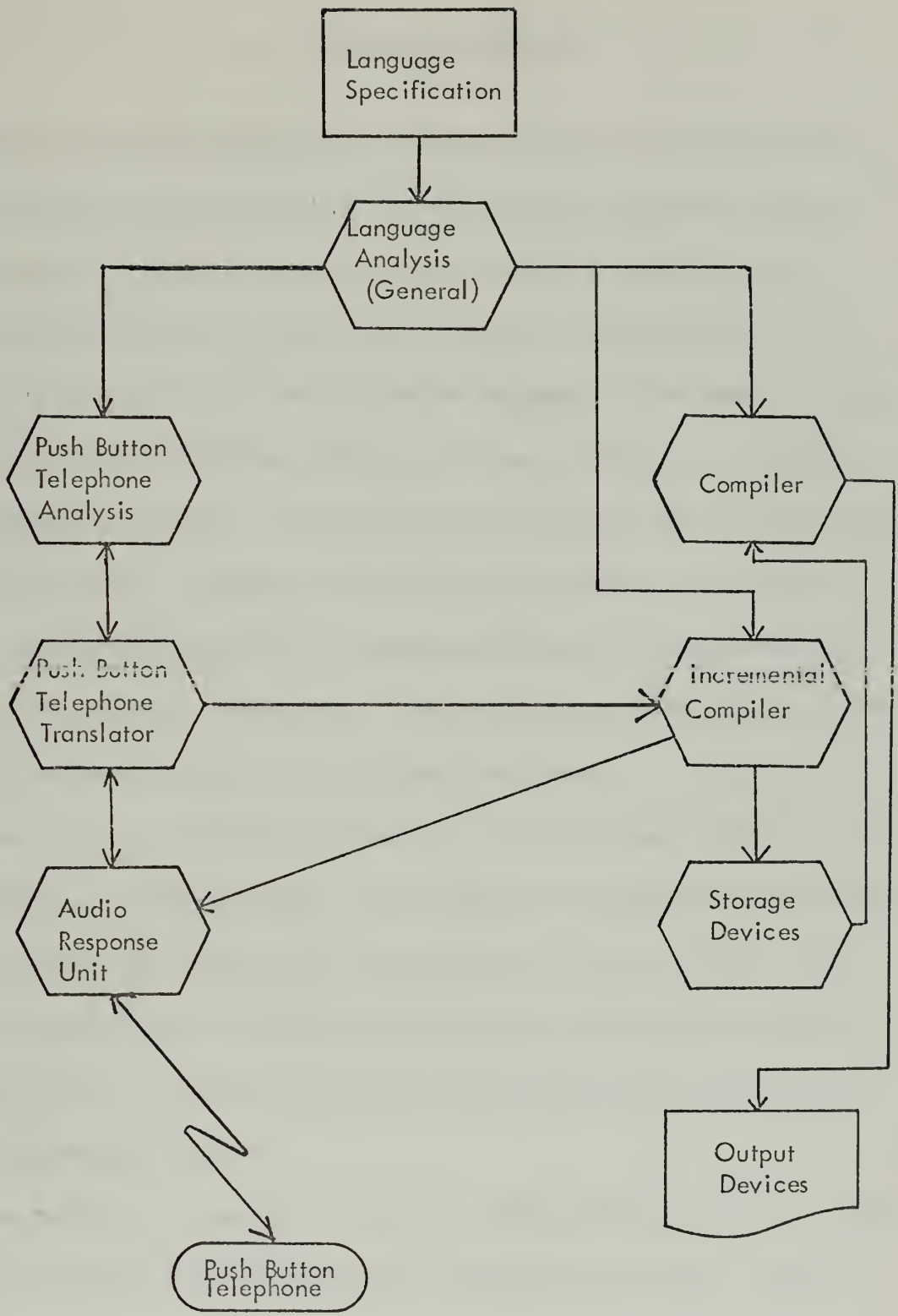

Language Specification
Language Analysis (General)
Push Button Telephone Analysis
Compiler
Push Button Telephone Translator
Incremental Compiler
Audio Response Unit
Storage Devices
Output Devices
Push Button Telephone

FIGURE 2

II. GENERAL DISCUSSION

The description of the computing system given in Figure 2 includes the push button telephone translating system at a level above the programming language it is processing. The output from the language analysis is corrected prior to considering the problems of the push button telephone coding scheme and is used in building the compiler (or translator) for the language. At this level, the output from the push button telephone translator is the source statement of the programming language in question. This output must be channeled to the language compiler for final processing. It appears that the two analysis steps could be combined to perform an analysis directed at the language problems and the push button telephone coding problems simultaneously. This would lead to a combined translator-compiler and would produce a more efficient total system.

There is also a possibility of additional use of the language analysis to improve on the push button coding scheme. The decoding of a statement such as "GO TO" might automatically indicate that a line number must be the next input. The system could then go directly into a number decoding scheme without a special number indicator. This sort of refinement would depend heavily upon the particular programming language.

The problems of the push button telephone coding scheme are of prime concern when considering a programming language specification procedure. A major problem area of a push button telephone coding scheme lies in the ambiguous transmission associated with each digital input. To maintain an average code

word length as small as possible, a method to determine that a "2" is not an "A" or "B" or "C" is necessary. The context of the programming language provides such a method. Also the context of a particular character in a terminal part of a language must be considered for the push button telephone coding scheme. To illustrate, a particular coding scheme could produce the code "367" for the word "FOR" and code "363" for the word "END". The "36" taken in context with the "7" or "3" would provide the necessary information for proper decoding.

The method used to specify a particular programming language obviously plays an important part in all the analysis procedures for the language. A syntactic description of a language provides a good method to examine the context of the language and also to determine the terminal parts of the language. Syntax examination therefore provides a good method for push button telephone coding analysis. Although no exhaustive study was performed, it appears that the attempt to make this sort of analysis using any other type of language specification would prove exceedingly difficult.

Coding of character strings (comments, headings, print strings, etc.) for a push button telephone system requires a special coding system within the program language coding system. This system must allow for the coding of the full alphabet, as well as special characters and numbers. The numbers and special characters can be decoded normally, but the alphabetical characters must be recognized individually since no contextual information would be available. A likely method for this operation is to transfer control to a special decoding section for character string decoding. The transfer would be caused by recognition of a special character, and the special decoding scheme would be specified by the user.

There are many of these special decoding schemes, but those that use the telephone key board to aid in remembering the code would appear to be the preferred systems.

Inherent to the push button telephone coding scheme are certain special codes which the user must define and input to the analyzer. These are parts of the coding scheme that are considered necessary to any system and must receive special consideration in the analysis and the translation phases. An example would be the "space designation", which is usually a part of any programming language. Since the "space bar" would not be available on a push button telephone, a special code must represent the space in the programming language. This is in line with the overall concept, as the transmission of a space over the telephone must be accomplished by some sequence of characters. Additional features such as an end-of-line designation or re-start capability would also be required and could also be defined as special parts of the coding scheme.

The variable introduces a unique coding problem. A full alphabetic coding scheme could be used to allow complete generality of variable coding. Where this flexibility in variables is not required, a restricted system is much more efficient. A variable designation parameter would provide such a system. One designation would be used to denote all variables, and the variables would be distinguished by their assigned number. An obvious choice for the variable designation parameter would be the letter "V". This would allow the assignment of variables in the sequence V1, V2, V3....Vn, where n is restricted only by the computer the translator will run on.

The input scanning operation of a working translator could operate character-by-character or be keyed to some other character set determination. The character-by-character scanning would require that all prefixes be eliminated from the programming language. This would be necessary to insure that a word such as "GO" (code "46") would not be decoded from the word "INT" (code "468"). This restriction could prove quite severe in some languages. The proper use of the characters "*" and "#" eliminates this problem. These characters can be used in the programming language to break the input number codes into subsets for decoding while serving as code characters for parts of the programming language. This approach appears to be the most fruitful.

Response for the pushbutton telephone system could be handled through different paths (Fig. 2) depending upon the response required. The loop from the telephone through the translator to the incremental compiler, and back, would provide the "desk calculator" service mentioned earlier. Line-by-line execution and even short program execution would be handled by this loop. Response for error detection could be between the telephone and the translator for errors in the push button telephone coding. An error in the programming language might not be detectable by the push button telephone translator. The incremental compiler would detect the error and initiate the error message. For a longer program with delayed output, a direct translation, store and input to job stream could be provided, but would not take full advantage of the system. The incremental compiler in the sequence would provide immediate error correction on a large percentage of the program errors. In this respect, the push button telephone system could increase the efficiency of the total computing facility.

III. TELE-CODER

The Tele-Coder is a system which analyses a programming language for acceptability of input through a push button telephone and builds a translator to decode the input. There are two main sections to the system, the analyzer and the translator, which are both written in STANFORD ALGOL W (Version 21 NOV1969). The analyzer reads a grammar in Backus Naur Form [2] (BNF) and analyses it for problems in coding the language into a set of twelve characters. The analyzer output is then used to build a translator for the language. Figure 3 is a flow chart showing the use of the Tele-Coder.

The Tele-Coder will analyze any context-free language. Any context-free grammar can be written in BNF and this is a requirement for the input to the analyzer.[3] This type of input was chosen because it covers a large class of languages and provides an explicit description of the language in compact form. Also, this method of describing a language has found widespread use in recent years and is familiar to most computer scientists.

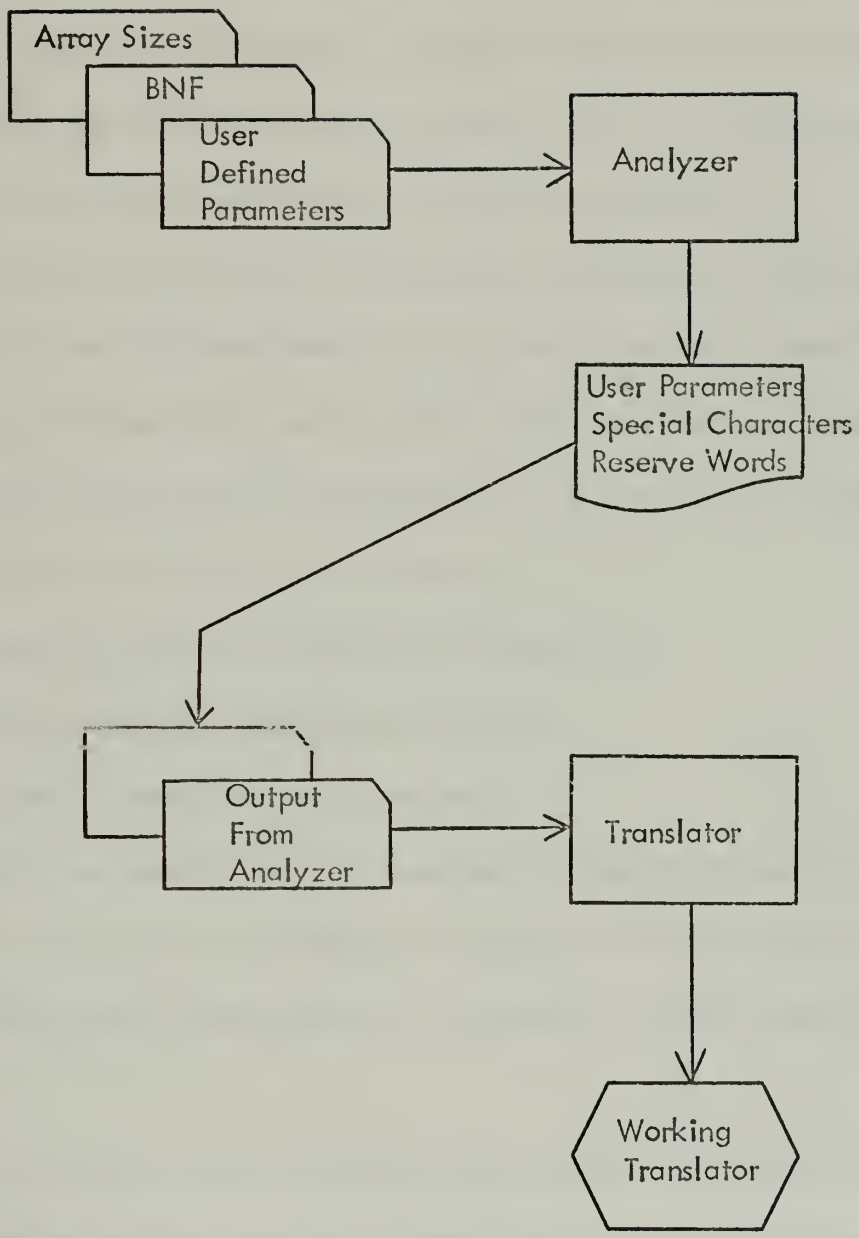
A. THE ANALYZER

A sample input to the analyzer is shown in appendix A. There are three main parts to the input; the array sizes, the language syntax and the user defined parameters.

---

[2] Also called Backus Normal Form.

[3] A good illustration of the use of BNF to specify a context-free grammar is given in[Ref. 7].



Flow Chart for use of Tele-Coder

Figure 3.

There are two array size numbers required, and these are used to dynamically allocate space for storage of the user defined parameters and the phrases of the grammar. The first of these numbers is used for the grammar and does not have to be exact but must be greater than the number of phrases in the grammar. The second number is the exact number of user defined parameters.

The grammar input follows a form similar to that used for the XPL system [Ref. 7] . The BNF metasymbols ::=, |, <, and > are used to break the productions into left parts followed by a list of right parts. The input to the analyzer does not require the use of the metasymbol |, or the use of the symbol ::=. The only restrictions on the input are as follows:

1. There must be only one production per input card.
2. There must be only one right part per card.
3. The left part must start in column one.

A left part is not required for every input card. When the left part is omitted, the input is assumed to be a right part alternative to the most recent left part. The grammar input is terminated by a card containing "EOG" beginning in column one.

There are seven required user defined parameters. These form the final input to the analyzer and they must be in a strict format and sequence. (See appendix A.) None may exceed five characters in length and codes for special characters must be two characters. Additionally, the special character represented must be included in the fifth column of the input string. The input strings must also be enclosed in double quotation marks (i.e. "). The input ordering of the seven required parameters is as follows:

1. space designation
2. number designation
3. variable designation
4. end of line designation
5. character string designation
6. quote string designation
7. re-start designation

The distinction between the character string and quote string designation is that the character string refers to program comment strings and the quote string is used for data type "character strings". If either designation is not required, "NO" may be input for parameter five and/or six.

In analysis of the grammar, special characters are first determined. These include any characters which are not members of the alphabet or the integers. They are assigned a code in the sequence *0, *1,...., *9, #0, #1,....,#9, **, *#, ##, #* and stored for output to be used by the translator. Codes assigned in the user defined parameters section are checked to assure that no conflicting assignments are made. The number of special character codes required may not exceed twenty-four. The total number of special character codes assigned is also output for use by the translator.

After completion of the special character coding, reserve words are determined and stored for coding. Reserve words are terminal parts of the programming language in question, and must be uniquely recognizable in the decoding process. The coding of these words consists of the mapping of the letters of the word onto the key numbers of the push button telephone keyboard. Conflicts may appear in this coding and they must be resolved by the user. For example, the words "IF" and "GE" would produce the number code "43". This problem could be resolved by using the code "GEQ" in place of "GE". The translator would then recognize

code "430" ("GEQ") as "GE" so the original language would be preserved. The number of reserve words is also tabulated for later use by the translator.

The output from the user defined parameter section, the special character section, and the reserve word section combine to make up the total input to the translator. The analyzer presently provides a printed output which must be prepared for input to the translator. The provision of a punched card output from the analyzer would speed up the total process but was not considered necessary to the study.

B. THE TRANSLATOR

The function of the translator is to accept information from the analyzer and use this to build a decoder for the programming language processed. A sample input to the translator is shown in appendix B.

The input to the translator also requires two array upper bounds. The first of these is the number of special characters, and this is followed by the number of reserve words. These numbers provide for dynamically allocated storage in the translator. The additional input consists of the user defined parameters, the reserve words and their coding, and the special characters and their coding. With the full input, the translator becomes a working model and is ready to accept coded input from a push button telephone.

The input scanning of the translator depends heavily upon the characters "*" and "#". These are used to break the input number codes into subsets for decoding. The use of these characters in forming the special character codes places them in the programming language where they may be used to serve this purpose with no

extra effort. Where two number codes fall adjacent, the space designation must be used to provide the necessary break.

The variable designation is not decoded to the specific input character by the translator but instead always decodes to the letter "V". This is not a requirement but was chosen for convenience. This method also serves to illustrate the efficient use of a digit that is not used as the beginning of a reserve word code. In one language used in testing, no reserve words began with the letters "Q" or "Z". This allowed the use of the digit "O" as the variable designation and so a two character code was not required. Since the digit "1" is not associated with another character or letter, it is always available for this type of use.

The character string and quote string options in the translator are dependent on a full alphabet coding scheme designed to illustrate this type of coding. The scheme is based on the alphabetic character location on the push buttons.[4] The code requires two characters per alphabetic character. The first character is simply the push button number on which the alphabetic character appears. The second character is determined by the relative position of the alphabetic character on the push button. The alphabetic character will appear in the "first", "middle" or "last" position. (See figure 1.) This position and the row of the first code character determines the second character. The number "4" represents the "first" position for row two and the number "6" represents the "last" position for that row. For example, the code for the letter "L" is "56" and code for the letter "X" is "98". Letters "Q" and "Z" are assumed to appear on push button "O" and the "middle" position of this key is not used.

[4] This scheme is similar to the one given in Ref. 1.

Error analysis for the translator is handled primarily in one subroutine. When an un-decodable sequence of characters is encountered, a message particular to the area where decoding was in process is first output. Control is then transferred to the error handling routine and the characters successfully decoded to the point of the error are output. Analysis to determine the sequence of non-decodable characters is also output. Finally, the line of code containing the error is discarded and control returned to the main program.

C. TESTING

The implementation of the Tele-Coder was directed at demonstrating the theory of such a system and to allow study of the various aspects of the system. Testing of a specific concept was considered adequate when a valid result was obtained, and no attempt was made to optimize the procedure in question. Additional testing and a period of evolution would be required to produce a good production model of the system.

Two programming languages were used to test the analysis portion of the Tele-Coder. The two languages were BASIC and BALGOL, and were chosen because they represent a significant programming capability and provided a good overall test of the system. The description of the syntax for BASIC was taken from Ref. 8. The syntax specification for BALGOL, which is a version of a student language written using the XPL system, was available from the writing of the language. In both cases, the changes required to make the language acceptable for input over a push button telephone were minor and there was ample room for expansion, should more powerful language be considered. The sample output from the analyzer included in this paper is from the analysis of BASIC.

The output from the analysis of BASIC was used to build and test the translator. The telephone input to test the translator was simulated by standard punch-card input, and the result of this testing is shown as the output from the translator. The significant features of the system are demonstrated by this output. A comparison of some of the features can give insight into some of the concepts of the system. The character string feature (output line 1) is a good example of the inefficiency of a coding scheme using no contextual information. Output line four uses context to reduce the average code word length. This reduction is obvious and significant when the two lines are compared. The advantageous segmantation of the code by the special characters is also quite obvious. The final lines of output were used to test and demonstrate the error detecting capabilities of the translator.

IV. CONCLUSION

The Analyzer has demonstrated the feasibility of a system for analysis and coding of certain programming languages for use with a push button telephone. The Translator has demonstrated the use of this analysis to produce a translator for decoding the programming language. The Tele-Coder then is a system that could be developed into an aid that could greatly enhance the use of the push button telephone as a computer terminal.

APPENDIX A

```
**************************************
*                                    *
*                                    *
*          INPUT TO ANALYZER         *
*                                    *
*                                    *
**************************************
```

```
   185
   7
<PROGRAM SET>  ::= <PROGRAM>
    <PROGRAM SET>*<PROGRAM>
<PROGRAM>  ::=  <STATEMENT SET> <END STATEMENT>
<STATEMENT SET>  ::=  <STATEMENT>
     <STATEMENT SET> <STATEMENT>
<END STATEMENT>  ::=  <LINE OPTION>END
<STATEMENT>  ::=  <LINE OPTION> <BASIC STATEMENT>
          <EMPTY>
<LINE OPTION>  ::=  <INTEGER>
          <EMPTY>
<INTEGER>  ::=  <DIGIT>
          <INTEGER> <DIGIT>
<EMPTY>  ::=
<DATA LIST>  ::=  <DATA ELEMENT>
          <DATA LIST> , <DATA ELEMENT>
<ALPHA VARIABLE>  ::=  <VARIABLE ASSIGN>
          <ALPHA DATA>
<NUMERIC DATA>  ::=  <SIGNED NUMBER>
          <NUMBER>
<SIGNED NUMBER>  ::=  <SIGN> <NUMBER>
<SIGN>  ::=  +
          -
<NUMBER>  ::=  <INTEGER>
          <FRACTION>
          <DECIMAL NUMBER>
<FRACTION>  ::=  .<INTEGER>
<DECIMAL NUMBER>  ::=  <INTEGER> <FRACTION>
<ALPHA DATA>  ::=  '<STRING>'
<STRING>  ::=  <CHARACTER>
          <STRING> <CHARACTER>
<CHARACTER>  ::=  <DIGIT>
          <ALPHA CHARACTER>
          <SPECIAL CHARACTER>
          <EMPTY>
<DIGIT>  ::=  0
          1
          2
          3
          4
          5
          6
          7
          8
          9
<ALPHA CHARACTER>  ::=  A
          B
          C
          D
          E
          F
          G
          H
          I
          J
          K
          L
          M
```

```
          N
          O
          P
          Q
          R
          S
          T
          U
          V
          W
          X
          Y
          Z
<SPECIAL CHARACTER>  ::=  =
          +
          *
          /
          (
          )
          .
          ,
          $
          -
          '
<BASIC STATEMENT>  ::=  <REMARK>
          <LET>
          <READ>
          <DATA>
          <PRINT>
          <GO TO>
          <IF>
          <FOR>
          <NEXT>
          <DIMENSION>
          <GOSUB>
          <RETURN>
          <PAGE>
          <STOP>
          <RESTORE>
<REMARK>  ::=  REM<CHARACTER SET>
<CHARACTER SET>  ::=  <STRING>
          <CHARACTER SET> ' <CHARACTER SET>
<LET>  ::=  LET<LET STATEMENT>
<LET STATEMENT>  ::=  <LET ALPHA>
          <LET NUMERIC>
<LET ALPHA>  ::=  <ALPHA ASSIGN> = <ALPHA EXPRESSION>
<ALPHA ASSIGN>  ::=  <ALPHA VARIABLE>
          <ALPHA ASSIGN> = <ALPHA VARIABLE>
<ALPHA EXPRESSION>  ::= <ALPHA DATA>
          <ALPHA VARIABLE>
<LET NUMERIC>  ::=  <VARIABLE ASSIGN> = <EXPRESSION>
<VARIABLE ASSIGN>  ::=  <VARIABLE>
          <VARIABLE ASSIGN> = <VARIABLE>
<VARIABLE>  ::=  <SIMPLE VARIABLE>
          <SUBSCRIPTED VARIABLE>
<SIMPLE VARIABLE>  ::=  <ALPHA CHARACTER>
          <ALPHA CHARACTER> <DIGIT>
<SUBSCRIPTED VARIABLE>  ::= <ALPHA CHARACTER>(<SUBSCRIPT LIS
<SUBSCRIPT LIST>  ::=  <EXPRESSION>
          <EXPRESSION> , <EXPRESSION>
<DATA ELEMENT>  ::=  <NUMERIC DATA>
<EXPRESSION>  ::=  <MULTIPLY FACTOR>
          <SIGN> <EXPRESSION>
          <EXPRESSION> <SIGN> <MULTIPLY FACTOR>
<MULTIPLY FACTOR>  ::=  <INVOLUTION FACTOR>
          <MULTIPLY FACTOR> <MULTIPLY SIGN> <INVOLUTION FACTO
<INVOLUTION FACTOR>  ::=  <TERM>
          <TERM>**<TERM>
<MULTIPLY SIGN>  ::=  *
          /
<TERM>  ::=  <NUMBER>
          <VARIABLE>
```

```
          (<EXPRESSION>)
          <FUNCTION TERM>
<FUNCTION TERM>  ::=  <FUNCTION>(<EXPRESSION>)
<FUNCTION>  ::=  SIN
          COS
          TAN
          ATN
          EXP
          ABS
          LOG
          SQR
          INT
          RND
<READ>  ::=  READ<READ LIST>
<READ LIST>  ::=  <READ VARIABLE>
          <READ LIST> ,<READ VARIABLE>
<READ VARIABLE>  ::=  <ALPHA VARIABLE>
          <VARIABLE>
<DATA>  ::=  DATA<DATA LIST>
<PRINT>  ::=  PRINT<PRINT LIST>
<PRINT LIST>  ::=  <PRINT ELEMENT>
          <PRINT LIST> ,<PRINT ELEMENT>
<PRINT ELEMENT>  ::=  <EMPTY>
          <ALPHA VARIABLE>
          <VARIABLE>
          <EXPRESSION>
          <ALPHA DATA>
          TAB(<EXPRESSION>)
<GO TO>  ::=  GO TO<INTEGER>
<IF>  ::=  IF<EXPRESSION> <RELATION> <EXPRESSION> THEN <INTE
<RELATION>  ::=  GT
          GE
          LT
          LE
          NE
          =
<FOR>  ::=  FOR<SIMPLE VARIABLE> = <EXPRESSION> TO <EXPRESS
<STEP OPTION>  ::=  <EMPTY>
          STEP<EXPRESSION>
<NEXT>  ::=  NEXT<SIMPLE VARIABLE>
<DIMENSION>  ::=  DIM<DIMENSION LIST>
<DIMENSION LIST>  ::=  <DIMENSION VARIABLE>
          <DIMENSION LIST> , <DIMENSION VARIABLE>
<DIMENSION VARIABLE>  ::=  <ALPHA CHARACTER>(<EXTENT>)
<EXTENT>  ::=  <INTEGER>
          <INTEGER> ,<INTEGER>
<GOSUB>  ::=  GOSUB<INTEGER>
<RETURN>  ::=  RETURN
<STOP>  ::=  STOP
<PAGE>  ::=  PAGE
<RESTORE>  ::=  RESTORE
          RESTORE$
EOG
"**    "
"1     "
"0     "
"##    "
"ALPHA"
"*5   '"
"#*    "
```

APPENDIX B

```
**********************************************
*                                            *
*                                            *
*          INPUT TO TRANSLATOR               *
*                                            *
*                                            *
**********************************************
```

```
   11
   35
"**    "
"##    "
"1     "
"0     "
"25742"
"*5    "
"#*    "
ALPHA
25742
END
363
REM
736
LET
538
SIN
746
COS
267
TAN
826
ATN
286
EXP
397
ABS
227
LOG
564
SQR
7C7
INT
468
RND
763
READ
7323
DATA
3282
PRINT
77468
TAB
822
GO
46
IF
43
THEN
8436
GT
48
GE
430
LT
58
LE
53
```

```
NE
63
FOR
367
TO
86
STEP
7837
NEXT
6398
DIM
346
GOSUB
46782
RETURN
738876
STOP
7867
PAGE
7243
RESTORE
7378673
"**0"
",*1"
"+*2"
"-*3"
".*4"
"!*5"
"=*6"
"/*7"
"(*8"
")*9"
"$#0"
```

OUTPUT FROM ANALYZER

PRODUCTIONS IN BNF

```
**********************************************************************

⟨PROGRAM SET⟩ ::= ⟨PROGRAM⟩
                | ⟨PROGRAM SET⟩*⟨PROGRAM⟩

⟨PROGRAM⟩ ::= ⟨STATEMENT SET⟩ ⟨END STATEMENT⟩

⟨STATEMENT SET⟩ ::= ⟨STATEMENT⟩
                  | ⟨STATEMENT SET⟩ ⟨STATEMENT⟩

⟨END STATEMENT⟩ ::= ⟨LINE OPTION⟩END

⟨STATEMENT⟩ ::= ⟨LINE OPTION⟩ ⟨BASIC STATEMENT⟩
              | ⟨EMPTY⟩

⟨LINE OPTION⟩ ::= ⟨INTEGER⟩
                | ⟨EMPTY⟩

⟨INTEGER⟩ ::= ⟨DIGIT⟩
            | ⟨INTEGER⟩ ⟨DIGIT⟩

⟨EMPTY⟩ ::=

⟨DATA LIST⟩ ::= ⟨DATA ELEMENT⟩
              | ⟨DATA LIST⟩ , ⟨DATA ELEMENT⟩

⟨ALPHA VARIABLE⟩ ::= ⟨VARIABLE ASSIGN⟩
                   | ⟨ALPHA DATA⟩

⟨NUMERIC DATA⟩ ::= ⟨SIGNED NUMBER⟩
                 | ⟨NUMBER⟩

⟨SIGNED NUMBER⟩ ::= ⟨SIGN   NUMBER⟩

⟨SIGN⟩ ::= +
         | -

⟨NUMBER⟩ ::= ⟨INTEGER⟩
           | ⟨FRACTION⟩
           | ⟨DECIMAL NUMBER⟩
```

⟨FRACTION⟩ ::= . INTEGER⟩

⟨DECIMAL NUMBER⟩ ::= ⟨INTEGER⟩ ⟨FRACTION⟩

⟨ALPHA DATA⟩ ::= '⟨STRING⟩'

⟨STRING⟩ ::= ⟨CHARACTER⟩
| ⟨STRING⟩ ⟨CHARACTER⟩

⟨CHARACTER⟩ ::= ⟨DIGIT⟩
| ⟨ALPHA CHARACTER⟩
| ⟨SPECIAL CHARACTER⟩
| ⟨EMPTY⟩

⟨DIGIT⟩ ::= 0
| 1
| 2
| 3
| 4
| 5
| 6
| 7
| 8
| 9

⟨BASIC STATEMENT⟩ ::= ⟨REMARK⟩
| ⟨LET⟩
| ⟨READ⟩
| ⟨DATA⟩
| ⟨PRINT⟩
| ⟨GO TO⟩
| ⟨IF⟩
| ⟨FOR⟩
| ⟨NEXT⟩
| ⟨DIMENSION⟩
| ⟨GOSUB⟩
| ⟨RETURN⟩
| ⟨PAGE⟩
| ⟨STOP⟩
| ⟨RESTORE⟩

⟨REMARK⟩ ::= REM⟨CHARACTER SET⟩

⟨CHARACTER SET⟩ ::= ⟨STRING⟩
| ⟨CHARACTER SET⟩ ' ⟨CHARACTER SET⟩

⟨ALPHA CHARACTER ⟩ ::= A
| B
| C
| D
| E
| F
| G
| H
| I
| J
| K
| L
| M
| N
| O
| P
| Q
| R
| S
| T
| U
| V
| W
| X
| Y
| Z

⟨SPECIAL CHARACTER⟩ ::= =
| +
| *
| /
| (
|)
| .
| ,
| $
| -
| '

⟨LET⟩ ::= LET⟨LET STATEMENT⟩

⟨LET STATEMENT⟩ ::= ⟨LET ALPHA⟩
| ⟨LET NUMERIC⟩

⟨LET ALPHA⟩ ::= ⟨ALPHA ASSIGN⟩ = ⟨ALPHA EXPRESSION⟩

⟨ALPHA ASSIGN⟩ ::= ⟨ALPHA VARIABLE⟩
| ⟨ALPHA ASSIGN⟩ = ⟨ALPHA VARIABLE⟩

⟨ALPHA EXPRESSION⟩ ::= ⟨ALPHA DATA⟩
| ⟨ALPHA VARIABLE⟩

⟨LET NUMERIC⟩ ::= ⟨VARIABLE ASSIGN⟩ = ⟨EXPRESSION⟩

⟨VARIABLE ASSIGN⟩ ::= ⟨VARIABLE⟩
| ⟨VARIABLE ASSIGN⟩ = ⟨VARIABLE⟩

⟨VARIABLE⟩ ::= ⟨SIMPLE VARIABLE⟩
| ⟨SUBSCRIPTED VARIABLE⟩

⟨SIMPLE VARIABLE⟩ ::= ⟨ALPHA CHARACTER⟩
| ⟨ALPHA CHARACTER⟩ ⟨DIGIT⟩

⟨SUBSCRIPTED VARIABLE⟩ ::= ⟨ALPHA CHARACTER⟩ (⟨SUBSCRIPT LIST⟩)

⟨SUBSCRIPT LIST⟩ ::= ⟨EXPRESSION⟩
| ⟨EXPRESSION⟩ , ⟨EXPRESSION⟩

⟨DATA ELEMENT⟩ ::= ⟨NUMERIC DATA⟩

⟨EXPRESSION⟩ ::= ⟨MULTIPLY FACTOR⟩
| ⟨SIGN⟩ ⟨EXPRESSION⟩
| ⟨EXPRESSION⟩ ⟨SIGN⟩ ⟨MULTIPLY FACTOR⟩

⟨MULTIPLY FACTOR⟩ ::= ⟨INVOLUTION FACTOR⟩
| ⟨MULTIPLY FACTOR⟩ ⟨MULTIPLY SIGN⟩ ⟨INVOLUTION FACTOR⟩

⟨INVOLUTION FACTOR⟩ ::= ⟨TERM⟩
| ⟨TERM⟩**⟨TERM⟩

⟨MULTIPLY SIGN⟩ ::= *
| /

⟨TERM⟩ ::= ⟨NUMBER⟩
| ⟨VARIABLE⟩
| (⟨EXPRESSION⟩)
| ⟨FUNCTION TERM⟩

⟨FUNCTION TERM⟩ ::= ⟨FUNCTION⟩ (⟨EXPRESSION⟩)

```
⟨FUNCTION⟩ ::= SIN
             | COS
             | TAN
             | ATN
             | EXP
             | ABS
             | LOG
             | SQR
             | INT
             | RND

⟨READ⟩ ::= READ⟨READ LIST⟩

⟨READ LIST⟩ ::= ⟨READ VARIABLE⟩
              | ⟨READ LIST⟩ , ⟨READ VARIABLE⟩

⟨READ VARIABLE⟩ ::= ⟨ALPHA VARIABLE⟩
                  | ⟨VARIABLE⟩

⟨DATA⟩ ::= DATA⟨DATA LIST⟩

⟨PRINT⟩ ::= PRINT⟨PRINT LIST⟩

⟨PRINT LIST⟩ ::= ⟨PRINT ELEMENT⟩
               | ⟨PRINT LIST⟩ , ⟨PRINT ELEMENT⟩

⟨PRINT ELEMENT⟩ ::= ⟨EMPTY⟩
                  | ⟨ALPHA VARIABLE⟩
                  | ⟨VARIABLE⟩
                  | ⟨EXPRESSION⟩
                  | ⟨ALPHA DATA⟩
                  | TAB(⟨EXPRESSION⟩)

⟨GO TO⟩ ::= GO TO⟨INTEGER⟩

⟨IF⟩ ::= IF⟨EXPRESSION⟩ ⟨RELATION⟩ ⟨EXPRESSION⟩ THEN ⟨INTEGER

⟨RELATION⟩ ::= GT
             | GE
             | LT
             | LE
             | NE
             | =

⟨FOR⟩ ::= FOR ⟨SIMPLE VARIABLE⟩ =

⟨EXPRESSION⟩ TO ⟨EXPRESSION⟩ ⟨STEP OPTION⟩
```

⟨STEP OPTION⟩ ::= ⟨EMPTY⟩
| STEP⟨EXPRESSION⟩

⟨NEXT⟩ ::= NEXT⟨SIMPLE VARIABLE⟩

⟨DIMENSION⟩ ::= DIM⟨DIMENSION LIST⟩

⟨DIMENSION LIST⟩ ::= ⟨DIMENSION VARIABLE⟩
| ⟨DIMENSION LIST⟩ , ⟨DIMENSION VARIABLE⟩

⟨DIMENSION VARIABLE⟩ ::= ⟨ALPHA CHARACTER⟩ (⟨EXTENT⟩)

⟨EXTENT⟩ ::= ⟨INTEGER⟩
| ⟨INTEGER⟩ ,⟨INTEGER⟩

⟨GOSUB⟩ ::= GOSUB⟨INTEGER⟩

⟨RETURN⟩ ::= RETURN

⟨STOP⟩ ::= STOP

⟨PAGE⟩ ::= PAGE

⟨RESTORE⟩ ::= RESTORE
| RESTORE$

USER SPECIFIED PARAMETERS
****

SPACE DESIGNATION = **

NUMBER DESIGNATION = 1

VARIABLE DESIGNATION = 0

END OF LINE DESIGNATION = ##

CHARACTER STRING DESIGNATION = ALPHA

QUOTE STRING DESIGNATION = *5 '

RESTART DESIGNATION = #*

NUMBER OF PARAMETERS = 6

SPECIAL CHARACTER CODING

****

SPECIAL CHARACTER	CHARACTER CODE	
*	*0	
,	*1	
+	*2	
-	*3	
.	*4	
'	*5	USER DEFINED OPTION
=	*6	
/	*7	
(	*8	
)	*9	
$	*0	
	**	USER DEFINED OPTION
	##	USER DEFINED OPTION
	#*	USER DEFINED OPTION

NUMBER OF SPECIAL CHARACTERS = 14

NUMBER CODING OF RESERVE WORDS

****

RESERVE WORD	NUMBER CODE	
ALPHA	25742	USER DEFINED OPTION
END	363	
REM	736	
LET	538	
SIN	746	
COS	267	
TAN	826	
ATN	286	
EXP	397	
ABS	227	
LOG	564	
SQR	707	
INT	468	
RND	763	
READ	7323	
DATA	3282	
PRINT	77468	
TAB(	822	

CHARACTER (DOES NOT CODE

GO	46
IF	43
THEN	8436
GT	48
GE	43

NUMBER CODE CONFLICT

IF
43
GE
43

LT	58
LE	53
NE	63
FOR	367
TO	86
STEP	7837
NEXT	6398
DIM	346

```
GOSUB                    46782
RETURN                   738876
STOP                     7867
PAGE                     7243
RESTORE                  7378673
RESTORE$                 7378673

CHARACTER  $  DOES NOT CODE

NUMBER CODE CONFLICT

   RESTORE
   7378673
   RESTORE$
   7378673


NUMBER OF RESERVE WORDS  =                36
```

OUTPUT FROM TRANSLATOR

```
736**2574287454679**77786644782164**313264666579877821873279**87453225742##
REM THIS PROGRAM DEMONSTRATES THE                                          1

736**2574233667864218 7**6633**21**2221794623**7778644782164**25742##
REM FORMAT OF A BASIC PROGRAM                                              2

                                                                           3

1100**7323**01*102*103##
100 READ V1, V2, V3                                                        4

    538**04*602*003##
    LET V4=V2*V3                                                           5

    538**05*6*4114*004##
    LET V5=.14*V4                                                          6

    538**06*604*305##
    LET V6=V4-V5                                                           7

    43**01*6**02**8436**1200##
    IF V1= V2 THEN 200                                                     8

    77468**01*102*103*104##
    PRINT V1, V2, V3, V4                                                   9

    46**86**1100##
    GO TO 100                                                             10

    538**05*6#*
```

```
        77468*8*5874532**0*88668732**798778466544*5*9##
        PRINT('THE QUOTE STRING')                                    11

3283**11*113*4150*11100##
DATA 1,3.50,100                                                      12

1200**363##
200 END                                                              13


        538#805*6*4114*004##
---INVALID SPECIAL CHARACTER---
 OUTPUT TO THIS POINT
        LET
 SPECIAL CHARACTER #8 DOES NOT DECODE

        438**05*6*4114*004##
---INVALID RESERVE WORD---
 OUTPUT TO THIS POINT

WORD 438 DOES NOT DECODE

736**2574287254679**77786644782164**3132646665798778218732?9**87453225742##
---INVALID ALPHA CHARACTER---
 OUTPUT TO THIS POINT
REM T
 WORD 254679 DOES NOT DECODE

        77468*8*5674532**0*88668732**798778466544*5*9##
---INVALID QUOTE CHARACTER---
 OUTPUT TO THIS POINT
        PRINT('
 WORD 674532 DOES NOT DECODE

        77468*8*6874532**0*88668732**79877846654432*6##
---INVALID RESERVE WORD---
 OUTPUT TO THIS POINT
        PRINT(=
 WORD 874532 DOES NOT DECODE


EOP
```

```
          ****************************************
          *                                      *
          *                                      *
          *              ANALYZER                *
          *                                      *
          *                                      *
          ****************************************


BEGIN
   INTEGER I,J,K,L,M,N,SC,PNUM,RWN,X;
   STRING(130) PRINT;
   STRING(160) ARRAY R(0::30);
   STRING(160) ARRAY RW(0::50);
   STRING(80) SYNTAX;
   STRING(80) LEFTPART, RIGHTPART, SCODE;
   STRING(26) ALPHA,ALPHANUMBER;
   STRING(10) SNUM;
   STRING(3) ARRAY S(0::39);
   STRING(1) BLANK;
   LOGICAL MOREWORK, MOREPRODUCTIONS, TERMINAL;
   LOGICAL FIRST;

 COMMENT  ************************************************

             EXECUTION BEGINS HERE.

          ************************************************;

   ALPHA:=  "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
   ALPHANUMBER:= "22233344455566670778889990";
   MOREWORK:= TRUE;
   MOREPRODUCTIONS:= TRUE;
   BLANK:= " ";
   READ(N);
   READ(X);
   PNUM:=0;
   PRINT:= " ";
   SNUM:= "0123456789";
   WHILE MOREWORK DO
   BEGIN
        STRING(160) ARRAY PRODUCTION(1::N);
        STRING(5) ARRAY OPTION(0::X);

 COMMENT  ************************************************

             PROCEDURE OPTIONS READS USER DEFINED PARA-
             METERS AND STORES THEM IN ARRAY OPTION. SEVEN
             PARAMETERS MUST BE INPUT AND MORE MAY BE
             DEFINED BY THE USER. IF THE CHARACTER STRING
             OR QUOTE STRING PARAMETERS ARE NOT REQUIRED,
             "NO" MUST BE INPUT FOR PARAMETERS 5 AND 6.
             ALL INPUTS MUST BE LEFT JUSTIFIED TO COLUMN
             ONE AND ENCLOSED IN QUOTATION MARKS. PARA-
             METERS MUST BE FIVE OR LESS CHARACTERS IN
             LENGTH. USER DEFINED SPECIAL CHARACTERS MUST
             BE TWO CHARACTERS AND THE SPECIAL CHARACTER
             MUST BE INPUT AS THE FIFTH CHARACTER OF THE
             PARAMETER. THE INPUT ORDERING OF THE SEVEN
             REQUIRED PARAMETERS IS AS FOLLOWS:
                  (1) SPACE DESIGNATION
                  (2) NUMBER DESIGNATION
                  (3) VARIABLE DESIGNATION
                  (4) END OF LINE DESIGNATION
                  (5) CHARACTER STRING DESIGNATION
                  (6) QUOTE STRING DESIGNATION
                  (7) RESTART DESIGNATION

          ************************************************;

 PROCEDURE OPTIONS;
        BEGIN
```

```
            FOR I:= 0 UNTIL (X-1) DO READ(OPTION(I));
            WRITE(BLANK);
            WRITE(BLANK);
            WRITE("USER SPECIFIED PARAMETERS");
            WRITE("**********************************",
                  "**********************");
            WRITE(BLANK);
            WRITE("SPACE DESIGNATION = ",OPTION(0));
            WRITE(BLANK);
            WRITE("NUMBER DESIGNATION = ",OPTION(1));
            WRITE(BLANK);
            WRITE("VARIABLE DESIGNATION = ",OPTION(2));
            WRITE(BLANK);
            WRITE("END OF LINE DESIGNATION = ",OPTION(3));
            WRITE(BLANK);
            WRITE("CHARACTER STRING DESIGNATION = ",
                  OPTION(4));
            WRITE(BLANK);
            WRITE("QUOTE STRING DESIGNATION = ",
                 OPTION(5));
            WRITE(BLANK);
            WRITE("RESTART DESIGNATION = ",OPTION(6));
            WRITE(BLANK);
            WRITE(BLANK);
            WRITE("NUMBER OF PARAMETERS = ",X-1);
            END OPTIONS;

COMMENT  ************************************************

            PROCEDURE LOAD ACCEPTS THE SYNTAX READ IN BY
            THE MAIN PROGRAM AND LOADS IT INTO THE ARRAY
            PRODUCTION FOR USE BY THE OTHER SUBROUTINES.

          ************************************************;

PROCEDURE LOAD;
        BEGIN
            PNUM:= PNUM+1;
            PRODUCTION(PNUM):= " ";
            RIGHTPART:= " ";
            K:= -1;
            J:=0;
            IF SYNTAX(0|1) = "<" THEN
            BEGIN
                 LEFTPART:= " ";
                 WHILE SYNTAX(J|1) ¬= ">" DO
                 BEGIN
                      LEFTPART(J|1):= SYNTAX(J|1);
                      J:= J+1;
                      END;
                 LEFTPART(J|1):= ">";
                 WHILE SYNTAX(J|1) ¬= "=" DO J:= J+1;
                 J:= J+1;
                 WHILE SYNTAX(J|1) = " " DO
                 BEGIN
                      IF J < 79 THEN J:= J+1
                      ELSE SYNTAX(J|1):= "@";
                      END;
                 WHILE (J < 79) AND (K < 79) DO
                 BEGIN
                      K:= K+1;
                      RIGHTPART(K|1):= SYNTAX(J|1);
                      J:= J+1;
                      END;
                 END ELSE
            BEGIN
                 WHILE SYNTAX(J|1) = " " DO J:= J+1;
                 WHILE (J < 79) AND (K < 79) DO
                 BEGIN
                      K:= K+1;
                      RIGHTPART(K|1):= SYNTAX(J|1);
                      J:= J+1;
```

```
                    END;
               END;
          IF SYNTAX(79|1) = "@" THEN RIGHTPART:= " ";
          PRODUCTION(PNUM)(0|80):= LEFTPART;
          PRODUCTION(PNUM)(80|80):= RIGHTPART;
          END LOAD;

COMMENT   ********************************************

          PROCEDURE ECHO PRINTS THE SYNTAX IN A STANDARD
          FORM.

          ********************************************;

PROCEDURE ECHO(INTEGER VALUE I);
     BEGIN
          FOR J:= 1 UNTIL I DO
          BEGIN
               PRINT:= " ";
               K:=0;
               WHILE PRODUCTION(J)(K|1) ¬= ">" DO
               BEGIN
                    PRINT(K|1):= PRODUCTION(J)(K|1);
                    K:= K+1;
                    IF K > 79 THEN GO TO NEXT;
                    END;
               PRINT(K|8):= ">  ::=  ";
               K:= K+7;
               L:= K+7;
               N:= L;
               WHILE PRODUCTION(J)(K|1) = " " DO
               BEGIN
                         IF K < 159 THEN K:= K+1
                         ELSE PRODUCTION(J)(K|1):= "@";
                         END;
               IF PRODUCTION(J)(K|1) = "@" THEN
               BEGIN
                    PRODUCTION(J)(K|1):= " ";
                    K:= L-7;
                    END;
               FOR M:= K UNTIL  159  DO
               BEGIN
                    PRINT(L|1):= PRODUCTION(J)(M|1);
                    L:= L+1;
                    IF L > 129 THEN L:=L-1;
                    END;
               IF J = 1 THEN WRITE(PRINT) ELSE
                    IF PRODUCTION(J)(0|80) =
                       PRODUCTION(J-1)(0|80) THEN
                    BEGIN
                         K:=0;
                         WRITE(BLANK);
                         WHILE PRODUCTION(J)(K|1) = " "
                              DO K:= K+1;
                         FOR M:=0 UNTIL N-5 DO
                              WRITEON(BLANK);
                         WRITEON("|");
                         RIGHTPART:=
                              PRODUCTION(J)(80|80);
                         WRITEON(BLANK,BLANK,RIGHTPART);
                         END ELSE
                    BEGIN
                         WRITE(BLANK);
                         WRITE(PRINT);
                    END;
               NEXT:  END;
          END ECHO;

COMMENT   ********************************************

          PROCEDURE SPECIALCHARACTER CHECKS THE SYNTAX
          FOR SPECIAL CHARACTERS SUCH AS *,/, AND -. THE
```

```
          CHARACTERS ARE THEN CODED AND STORED IN THE
          ARRAY S.

          ************************************************;

PROCEDURE SPECIALCHARACTER(INTEGER VALUE I);
        BEGIN
             WRITE(BLANK);
             WRITE("SPECIAL CHARACTER           ",
                   "CHARACTER CODE");
             WRITE(BLANK);
             INTFIELDSIZE:= 1;
             FOR J:= 0 UNTIL 39 DO S(J):= " ";
             FOR J:= 0 UNTIL (X-1) DO
             BEGIN
                  IF (OPTION(J)(0|1) = "*")
                 OR (OPTION(J)(0|1) = "#") THEN
                   BEGIN
                        IF OPTION(J)(0|2) = "**" THEN
                           SC:=20;
                        IF OPTION(J)(0|2) = "*#" THEN
                           SC:=21;
                        IF OPTION(J)(0|2) = "##" THEN
                           SC:=22;
                        IF OPTION(J)(0|2) = "#*" THEN
                           SC:=23;
                        IF DECODE(OPTION(J)(1|1))>= 240 THEN
                        BEGIN
                             SC:= DECODE(OPTION(J)(1|1));
                             SC:= SC-240;
                             IF OPTION(J)(0|1) = "#" THEN
                                SC:= SC+10;
                             END;
                        S(SC)(1|2):= OPTION(J)(0|2);
                        S(SC)(0|1):= OPTION(J)(4|1);
                        END;
                   END;
             SC:= -1;
             FOR J:= 1 UNTIL I DO
             BEGIN
                  RIGHTPART:= PRODUCTION(J)(80|80);
                  FOR K:= 0 UNTIL 79 DO
                  BEGIN
                        TERMINAL:= TRUE;
                        IF RIGHTPART(K|1) = "<" THEN
                           TERMINAL:= FALSE;
                        IF RIGHTPART(K|1) = ">" THEN
                           TERMINAL:= TRUE;
                        IF ((DECODE(RIGHTPART(K|1))) < 193)
                       AND (RIGHTPART(K|1) ¬= ">")
                       AND (RIGHTPART(K|1) ¬= " ")
                       AND TERMINAL THEN
                        BEGIN
                             FOR M:= 0 UNTIL 39 DO
                                  IF S(M)(0|1) =
                                       RIGHTPART(K|1) THEN
                                            GO TO SKIP;
                             SC:= SC+1;
                             WHILE  S(SC) ¬= "   " DO
                             BEGIN
                                  WRITE("     ",S(SC)(0|1),
                                        "          ",
                                        "          ",
                                        S(SC)(1|2),"   ",
                                        "USER DEFINED ",
                                        "OPTION");
                                  SC:= SC+1;
                                  END;
                             S(SC)(0|1):= RIGHTPART(K|1);
                             IF SC < 10 THEN
                                  L:= SC ELSE L:= SC-10;
                             IF SC < 10 THEN
```

```
                              S(SC)(1|1):= "*" ELSE
                              S(SC)(1|1):= "#";
                         S(SC)(2|1):= SNUM(L|1);
                         WRITE("     ",S(SC)(0|1),
                               "                    ",
                               "     ",S(SC)(1|2));
                         FOR M:= 0 UNTIL (X-1) DO
                         IF S(SC)(1|2) = OPTION(M)(0|2)
                         THEN
                              WRITEON("   USER DEFINED",
                                   " OPTION");
                         IF SC > 21 THEN
                              WRITE("TOO MANY SPECIAL",
                                   "CHARACTERS");
                         SKIP:  END;
                    END;
               END;
          K:= SC;
          FOR M:= (SC+1) UNTIL 38 DO
          BEGIN
               IF S(M) ¬= "   " THEN
               BEGIN
                    WRITE("     ",S(M)(0|1),"          ",
                          "                ",
                           S(M)(1|2),"   ",
                           "USER DEFINED OPTION");
                    K:= K+1;
                    END;
               END;
          K:= K+1;
          INTFIELDSIZE:= 14;
          WRITE(BLANK);
          WRITE(BLANK);
          WRITE("NUMBER OF SPECIAL CHARACTERS = ",K);
          END SPECIALCHARACTER;

COMMENT  ***************************************************

          PROCEDURE RESERVEWORD LOCATES RESERVE WORDS
          IN THE LANGUAGE AND STORES THEM IN ARRAY RW.

         ***************************************************;

PROCEDURE RESERVEWORD(INTEGER VALUE I);
       BEGIN
          RWN:=0;
          FOR J:= 0 UNTIL X DO
          BEGIN
               IF (DECODE(OPTION(J)(0|1)) >= 193)
              AND (DECODE(OPTION(J)(0|1)) <= 233)
              AND (OPTION(J) ¬= "NO   ") THEN
               BEGIN
                    RW(RWN):= OPTION(J);
                    RWN:=RWN+1;
                    END;
               END;
          FOR J:= 1 UNTIL I DO
          BEGIN
               RIGHTPART:= PRODUCTION(J)(80|80);
               K:=0;
               TERMINAL:= TRUE;
               WHILE K < 79 DO
               BEGIN
                    IF RIGHTPART(K|1) = "<" THEN
                         TERMINAL:= FALSE;
                    IF RIGHTPART(K|1) = ">" THEN
                         TERMINAL:= TRUE;
                    SCODE:= " ";
                    SC:=0;
                    WHILE ((((DECODE(RIGHTPART(K|1)))
                         >= 193)
                   AND ((DECODE(RIGHTPART(K|1)))
```

```
                        <= 233))
                    OR (SC > 0))
                    AND TERMINAL DO
                     BEGIN
                          SCODE(SC|1):= RIGHTPART(K|1);
                          SC:= SC+1;
                          K:= K+1;
                          IF RIGHTPART(K|1) = "<" THEN
                             TERMINAL:= FALSE;
                          IF RIGHTPART(K|1) = " " THEN
                             TERMINAL:= FALSE;
                          IF K > 79 THEN
                          BEGIN
                               K:=79;
                               TERMINAL:= FALSE;
                               END;
                          END;
                    IF SC > 1 THEN
                    BEGIN
                          RW(RWN):= SCODE;
                          FOR L:=  0 UNTIL (RWN-1) DO
                               IF RW(RWN) = RW(L) THEN
                                  RWN:= RWN-1;
                          RWN:= RWN+1;
                          END;
                    K:= K+1;
                    END;
               END;
          END RESERVEWORD;

COMMENT   ***************************************************

          PROCEDURE WORDNUMBER ASSIGNS THE NUMBER CODE
          TO THE RESERVE WORDS AND CHECKS FOR CONFLICT-
          ING NUMBER CODES.

          ***************************************************;

PROCEDURE WORDNUMBER(INTEGER I);
       FOR J:= 0 UNTIL I-1 DO
       BEGIN
            WRITE("       ");
            K:= 0;
            WHILE RW(J)(K|1) ¬= " " DO
            BEGIN
                 WRITEON(RW(J)(K|1));
                 K:= K+1;
                 END;
            M:= K;
            WHILE M < 20 DO
            BEGIN
                 WRITEON(" ");
                 M:= M+1;
                 END;
            FOR L:= 0 UNTIL (K-1) DO
            BEGIN
                 M:=0;
                 WHILE RW(J)(L|1) ¬= ALPHA(M|1) DO
                 BEGIN
                      M:= M+1;
                      IF M > 25 THEN
                      BEGIN
                           WRITE(BLANK);
                           WRITE("CHARACTER  ",RW(J)(L|1),
                                 "  DOES NOT CODE");
                           WRITE(BLANK);
                           GO TO HOP;
                           END;
                      END;
                 RW(J)(L+80|1):= ALPHANUMBER(M|1);
                 WRITEON(ALPHANUMBER(M|1));
                 HOP:  END;
```

```
            FOR N:= 0 UNTIL (X-1) DO
                 IF RW(J)(0|5) = OPTION(N) THEN
                      WRITEON("   USER DEFINED OPTION");
            FOR N:= 0 UNTIL (J-1) DO
            BEGIN
                 IF RW(N)(80|80) = RW(J)(80|80) THEN
                 BEGIN
                      WRITE(BLANK);
                      WRITE("NUMBER CODE CONFLICT");
                      WRITE(BLANK);
                      WRITE(BLANK);
                      K:=0;
                      WHILE RW(N)(K|1) ¬= " " DO
                      BEGIN
                           WRITEON(RW(N)(K|1));
                           K:= K+1;
                           END;
                      WRITE(BLANK);
                      K:=0;
                      WHILE RW(N)(80+K|1) ¬= " " DO
                      BEGIN
                           WRITEON(RW(N)(80+K|1));
                           K:= K+1;
                           END;
                      WRITE(BLANK);
                      K:=0;
                      WHILE RW(J)(K|1) ¬= " " DO
                      BEGIN
                           WRITEON(RW(J)(K|1));
                           K:= K+1;
                           END;
                      WRITE(BLANK);
                      K:=0;
                      WHILE RW(J)(80+K|1) ¬= " " DO
                      BEGIN
                           WRITEON(RW(J)(80+K|1));
                           K:= K+1;
                           END;
                      WRITE(BLANK);
                      WRITE(BLANK);
                      END;
                 END;
            END WORDNUMBER;

COMMENT   ************************************************

            PROCEDURE ANALYZE CALLS SUBROUTINES TO PER-
            FORM THE ANALYSIS OF THE SYNTAX.

          ************************************************;

PROCEDURE ANALYZE;
        BEGIN
            WRITE(BLANK);
            WRITE(BLANK);
            WRITE("PRODUCTIONS IN BNF");
            WRITE("**********************************",
                  "**********************");
            WRITE(BLANK);
            ECHO(PNUM);
            OPTIONS;
            WRITE(BLANK);
            WRITE(BLANK);
            WRITE(BLANK);
            WRITE("SPECIAL CHARACTER CODING");
            WRITE("**********************************",
                  "**********************");
            SPECIALCHARACTER(PNUM);
            RESEPVEWORD(PNUM);
            WRITE(BLANK);
            WRITE(BLANK);
            WRITE("NUMBER CODING OF RESERVE WORDS");
```

```
            WRITE("*************************************",
                  "**********************");
            WRITE(BLANK);
            WRITE("RESERVE WORD           NUMBER CODE");
            WRITE(BLANK);
            WORDNUMBER(RWN);
            RWN:= RWN-1;
            WRITE(BLANK);
            WRITE("NUMBER OF RESERVE WORDS = ",RWN);
            END ANALYZE;

COMMENT     ***************************************************

            MAIN PROGRAM BEGINS HERE.

            ***************************************************;

        WHILE MOREPRODUCTIONS DO
        BEGIN
            READCARD(SYNTAX);
            IF SYNTAX(0|3) = "EOG" THEN
            BEGIN
                MOREWORK:= FALSE;
                MOREPRODUCTIONS:= FALSE;
                ANALYZE;
                END ELSE
            LOAD;
            END MOREPRODUCTIONS;
        END MOREWORK;
END.
```

```
        ****************************************
        *                                      *
        *                                      *
        *              TRANSLATOR              *
        *                                      *
        *                                      *
        ****************************************


BEGIN
   INTEGER I,J,K,L,M,SC,RWN,LINE;
   INTEGER AL, EC, NU, SP, VA, QA, RE;
   STRING(80) INPUT, OUTPUT, WORD, HOLD;
   STRING(52) ALPHANUMBER;
   STRING(26) ABC;
   STRING(6) ALPHA, EOL, NUMB, SPACE, VAR, SPWORD, QALPHA;
   STRING(6) RESTART;
   STRING(2) SWORD;
   STRING(1) BLANK;
   LOGICAL ERROR;


 COMMENT  ************************************************
             EXECUTION BEGINS HERE.

          ************************************************;

   READ(SC);
   READ(RWN);
   BLANK:= " ";
   ERROR:= FALSE;
   ALPHANUMBER:= "212223313233444546545556646566777879";
   ALPHANUMBER(36|16):= "8788899793990*0#";
   ABC:= "ABCDEFGHIJKLMNOPRSTUVWXYQZ";
     BEGIN
        STRING(160) ARRAY RW(0::RWN);
        STRING(3) ARRAY S(0::SC);

 COMMENT  ************************************************

             PROCEDURE ERROW PRINTS OUTPUT ERROR MESSAGES
             AND POINTS OUT THE ERROR LOCATION IN THE INPUT
             STREAM.

          ************************************************;

 PROCEDURE ERROW(INTEGER VALUE A);
         BEGIN
              LINE:= LINE-1;
              ERROR:= TRUE;
              WRITE(" OUTPUT TO THIS POINT");
              WRITE(OUTPUT);
              IF (INPUT(A|1) ¬= "*")
             AND (INPUT(A|1) ¬= "#") THEN
              BEGIN
                   HOLD:= " ";
                   L:=0;
                   WHILE DECODE(INPUT(A|1)) > 239 DO
                   BEGIN
                        HOLD(L|1):= INPUT(A|1);
                        L:= L+1;
                        A:= A+1;
                        END;
                   WRITE(" WORD ");
                   L:=0;
                   WHILE HOLD(L|1) ¬= " " DO
                   BEGIN
                        WRITEON(HOLD(L|1));
                        L:= L+1;
                        END;
                   WRITEON(" DOES NOT DECODE");
                   END
```

```
            ELSE
            BEGIN
                 WRITE(" SPECIAL CHARACTER ");
                 WRITEON(INPUT(A|2));
                 WRITEON(" DOES NOT DECODE");
                 END;
            END ERROW;

COMMENT  ***************************************************

            PROCEDURE OPTIONS READS IN USER DEFINED PARA-
            METERS AND DETERMINES THEIR CHARACTER LENGTH.

          ***************************************************;

PROCEDURE OPTIONS;
        BEGIN
            READ(SPACE);
            READ(EOL);
            READ(NUMB);
            READ(VAR);
            READ(ALPHA);
            READ(QALPHA);
            READ(RESTART);
            SP:=0;
            WHILE SPACE(SP|1) ¬= " " DO SP:= SP+1;
            SP:= SP-1;
            EO:=0;
            WHILE EOL(EO|1) ¬= " " DO EO:= EO+1;
            EO:= EO-1;
            NU:=0;
            WHILE NUMB(NU|1) ¬= " " DO NU:= NU+1;
            NU:= NU-1;
            VA:=0;
            WHILE VAR(VA|1) ¬= " " DO VA:= VA+1;
            VA:= VA-1;
            AL:=0;
            WHILE ALPHA(AL|1) ¬= " " DO AL:= AL+1;
            AL:= AL-1;
            QA:=0;
            WHILE QALPHA(QA|1) ¬= " " DO QA:= QA+1;
            QA:= QA-1;
            RE:=0;
            WHILE RESTART(RE|1) ¬= " "  DO RE:= RE+1;
            RE:= RE-1;
            END OPTIONS;

COMMENT  ***************************************************

            PROCEDURE SCAN2 DECODES RESERVED WORDS.

          ***************************************************;

PROCEDURE SCAN2;
        BEGIN
            WORD:= " ";
            L:=0;
            WHILE DECODE(INPUT(K|1)) > 240 DO
            BEGIN
                 WORD(L|1):= INPUT(K|1);
                 K:= K+1;
                 L:= L+1;
                 END;
            J:=0;
            WHILE J <= RWN DO
            BEGIN
                 IF WORD = RW(J)(80|80) THEN
                 BEGIN
                      HOLD:= " ";
                      M:=0;
                      WHILE RW(J)(M|1) ¬= " " DO
                      BEGIN
```

```
                        OUTPUT(I|1):= RW(J)(M|1);
                        HOLD:= RW(J)(M|1);
                        M:= M+1;
                        I:= I+1;
                        END;
                    GO TO NEXT;
                    END;
               J:= J+1;
               IF J > RWN THEN
               BEGIN
                    L:=0;
                    WHILE WORD(L|1) ¬= " " DO
                    BEGIN
                         L:= L+1;
                         K:= K-1;
                         END;
                    WRITE("---INVALID RESERVE WORD---");
                    ERROW(K);
                    K:=79;
                    END;
               END;
          NEXT:  END SCAN2;

COMMENT  ************************************************

          PROCEDURE SCAN1 DECODES SPECIAL CHARACTERS.

         ************************************************;

PROCEDURE SCAN1;
     BEGIN
          SWORD:= INPUT(K|2);
          L:=0;
          WHILE SWORD ¬= S(L)(1|2) DO
          BEGIN
               L:= L+1;
               IF L > SC THEN
               BEGIN
                    WRITE("---INVALID SPECIAL ",
                          "CHARACTER---");
                    ERROW(K);
                    K:=79;
                    GO TO NEXT;
                    END;
               END;
          OUTPUT(I|1):= S(L)(0|1);
          K:= K+2;
          I:= I+1;
     NEXT:  END SCAN1;

COMMENT  ************************************************

          PROCEDURE QALPHABET DECODES QUOTED STRINGS.
          USER DEFINED PARAMETERS SPACE AND NUMBER AND
          ALL SPECIAL CHARACTERS ARE DECODED AS THOUGH
          THEY WERE NOT QUOTED.

         ************************************************;

PROCEDURE QALPHABET;
     BEGIN
          NEXT:  IF K > 78 THEN GO TO FINIS;
          SPWORD:= " ";
          J:=0;
          L:=K;
          WHILE J <= QA DO
          BEGIN
               SPWORD(J|1):= INPUT(L|1);
               J:= J+1;
               L:= L+1;
               IF L > 78 THEN GO TO FINIS;
               END;
```

```
WHILE SPWORD ¬= QALPHA DO
BEGIN
      SPWORD:= " ";
      J:=0;
      L:= K;
      WHILE J <= SP DO
      BEGIN
            SPWORD(J|1):= INPUT(L|1);
            J:= J+1;
            L:= L+1;
            IF L > 78 THEN GO TO FINIS;
            END;
      IF SPWORD = SPACE THEN
      BEGIN
            K:= K+SP+1;
            I:= I+1;
            GO TO NEXT;
            END;
      SPWORD:= " ";
      J:=0;
      L:= K;
      WHILE J <= NU DO
      BEGIN
            SPWORD(J|1):= INPUT(L|1);
            J:= J+1;
            L:= L+1;
            IF L > 78 THEN GO TO FINIS;
            END;
      IF SPWORD = NUMB THEN
      BEGIN
            K:= K+1;
            WHILE DECODE(INPUT(K|1)) > 239 DO
            BEGIN
                  OUTPUT(I|1):= INPUT(K|1);
                  K:= K+1;
                  I:= I+1;
                  END;
            GO TO NEXT;
            END;
       IF(INPUT(K|1) = "*")
      OR (INPUT(K|1) = "#") THEN
       BEGIN
            SCAN1;
            GO TO NEXT;
            END;
       M:=0;
       L:=0;
       WHILE INPUT(K|2) ¬= ALPHANUMBER(L|2) DO
       BEGIN
            M:= M+1;
            L:= L+2;
            IF L = 52 THEN
            BEGIN
                  WRITE("---INVALID QUOTE ",
                        "CHARACTER---");
                  ERROW(K);
                  K:=79;
                  GO TO NEXT;
                  END;
            END;
       OUTPUT(I|1):= ABC(M|1);
       I:= I+1;
       K:= K+2;
       GO TO NEXT;
       END;
       IF SPWORD = QALPHA THEN
       BEGIN
            K:= K+QA+1;
            FOR J:= 0 UNTIL SC DO
            IF S(J)(1|2) = SPWORD(0|2) THEN
            BEGIN
                  OUTPUT(I|1):= S(J)(0|1);
```

```
                        I:= I+1;
                        END;
                    END;
            FINIS:  END QALPHABET;

COMMENT   ************************************************

            PROCEDURE ALPHABET DECODES CHARACTER STRINGS.
            USER DEFINED PARAMETERS SPACE AND NUMBER AND
            ALL SPECIAL CHARACTERS ARE DECODED AS THOUGH
            THEY WERE NOT CHARACTER STRINGS.

          ************************************************;

PROCEDURE ALPHABET;
        BEGIN
            NEXT:   IF K > 78 THEN GO TO FINIS;
            SPWORD:= " ";
            J:=0;
            L:=K;
            WHILE J <= AL DO
            BEGIN
                SPWORD(J|1):= INPUT(L|1);
                J:= J+1;
                L:= L+1;
                IF L > 78 THEN GO TO FINIS;
                END;
            WHILE SPWORD ¬= ALPHA DO
            BEGIN
                SPWORD:= " ";
                J:=0;
                L:= K;
                WHILE J <= SP DO
                BEGIN
                    SPWORD(J|1):= INPUT(L|1);
                    J:= J+1;
                    L:= L+1;
                    IF L > 78 THEN GO TO FINIS;
                    END;
                IF SPWORD = SPACE THEN
                BEGIN
                    K:= K+SP+1;
                    I:= I+1;
                    GO TO NEXT;
                    END;
                SPWORD:= " ";
                J:=0;
                L:= K;
                WHILE J <= NU DO
                BEGIN
                    SPWORD(J|1):= INPUT(L|1);
                    J:= J+1;
                    L:= L+1;
                    IF L > 78 THEN GO TO FINIS;
                    END;
                IF SPWORD = NUMB THEN
                BEGIN
                    K:= K+1;
                    WHILE DECODE(INPUT(K|1)) > 239 DO
                    BEGIN
                        OUTPUT(I|1):= INPUT(K|1);
                        K:= K+1;
                        I:= I+1;
                        END;
                    GO TO NEXT;
                    END;
                 IF(INPUT(K|1) = "*")
                OR (INPUT(K|1) = "#") THEN
                 BEGIN
                    SCAN1;
                    GO TO NEXT;
                    END;
```

```
                    M:=0;
                    L:=0;
                    WHILE INPUT(K|2) ¬= ALPHANUMBER(L|2) DO
                    BEGIN
                         M:= M+1;
                         L:= L+2;
                         IF L = 52 THEN
                         BEGIN
                              WRITE("---INVALID ALPHA ",
                                    "CHARACTER---");
                              ERROW(K);
                              K:=79;
                              GO TO NEXT;
                              END;
                         END;
                    OUTPUT(I|1):= ABC(M|1);
                    I:= I+1;
                    K:= K+2;
                    GO TO NEXT;
                    END;
                    IF SPWORD = ALPHA THEN
                    BEGIN
                         K:= K+AL+1;
                         FOR J:= 0 UNTIL SC DO
                         IF S(J)(1|2) = SPWORD(0|2) THEN
                         BEGIN
                              OUTPUT(I|1):= S(J)(0|1);
                              I:= I+1;
                              END;
                         END;
               FINIS: END ALPHABET;

COMMENT  ***************************************************

               PROCEDURE SCAN ACCEPTS A SINGLE LINE OF INPUT
               AND SCANS IT CHARACTER BY CHARACTER. USER
               DEFINED CHARACTERS AND OPTIONS ARE DECODED BY
               SCAN AND OTHER ROUTINES ARE CALLED TO PERFORM
               THE DECODING OF RESERVE WORDS AND SYNTAX
               DEFINED SPECIAL CHARACTERS.

             ***************************************************;

PROCEDURE SCAN;
          BEGIN
               OUTPUT:= " ";
               I:=0;
               K:=0;
               WHILE K < 79 DO
               BEGIN
                    IF INPUT(K|1) = " " THEN
                    BEGIN
                         K:= K+1;
                         IF K = 79 THEN GO TO NEXT;
                         I:=K;
                         END
                    ELSE
                    BEGIN
                      AGAIN:  SPWORD:= " ";
                         J:=0;
                         L:= K;
                         WHILE J <= SP DO
                         BEGIN
                              SPWORD(J|1):= INPUT(L|1);
                              J:= J+1;
                              L:= L+1;
                              IF L > 78 THEN GO TO NEXT;
                              END;
                         IF SPWORD = SPACE THEN
                         BEGIN
                              OUTPUT(I|1):= " ";
                              K:= K+SP+1;
```

```
      I:= I+1;
      GO TO AGAIN;
      END;
SPWORD:= " ";
J:=0;
L:= K;
WHILE J <= EO DO
BEGIN
      SPWORD(J|1):= INPUT(L|1);
      J:= J+1;
      L:= L+1;
      IF L > 78 THEN GO TC NEXT;
      END;
IF SPWORD = EOL THEN
BEGIN
      K:=79;
      GO TO NEXT;
      END;
SPWORD:= " ";
J:=0;
L:= K;
WHILE J <= NU DO
BEGIN
      SPWORD(J|1):= INPUT(L|1);
      J:= J+1;
      L:= L+1;
      IF L > 78 THEN GO TC NEXT;
      END;
IF SPWORD = NUMB THEN
BEGIN
      K:= K+1;
      WHILE DECODE(INPUT(K|1))>239 DO
      BEGIN
            OUTPUT(I|1):= INPUT(K|1);
            K:= K+1;
            I:= I+1;
            END;
      GO TO AGAIN;
      END;
SPWORD:= " ";
J:=0;
L:= K;
WHILE J <= VA DO
BEGIN
      SPWORD(J|1):= INPUT(L|1);
      J:= J+1;
      L:= L+1;
      IF L > 78 THEN GO TC NEXT;
      END;
IF SPWORD = VAR THEN
BEGIN
      OUTPUT(I|1):= "V";
      I:= I+1;
      K:= K+VA+1;
      WHILE DECODE(INPUT(K|1))>239 DO
      BEGIN
            OUTPUT(I|1):= INPUT(K|1);
            K:= K+1;
            I:= I+1;
            END;
      GO TO AGAIN;
      END;
SPWORD:= " ";
J:=0;
L:= K;
WHILE J <= AL DO
BEGIN
      SPWORD(J|1):= INPUT(L|1);
      J:= J+1;
      L:= L+1;
      IF L > 78 THEN GO TC NEXT;
      END;
```

```
                IF SPWORD = ALPHA THEN
                BEGIN
                     FOR J:= 0 UNTIL SC DO
                     IF S(J)(1|2) = SPWORD(0|2) THEN
                     BEGIN
                          OUTPUT(I|1):= S(J)(0|1);
                          I:= I+1;
                          END;
                     K:= K+AL+1;
                     ALPHABET;
                     IF K > 78 THEN
                          GO TO NEXT
                     ELSE
                          GO TO AGAIN;
                     END;
                SPWORD:= " ";
                J:=0;
                L:= K;
                WHILE J<= QA DO
                BEGIN
                     SPWORD(J|1):= INPUT(L|1);
                     J:= J+1;
                     L:= L+1;
                     IF L > 78 THEN GO TO NEXT;
                     END;
                IF SPWORD = QALPHA THEN
                BEGIN
                     FOR J:= 0 UNTIL SC DO
                     IF S(J)(1|2) = SPWORD(0|2) THEN
                     BEGIN
                          OUTPUT(I|1):= S(J)(0|1);
                          I:= I+1;
                          END;
                     K:= K+QA+1;
                     QALPHABET;
                     IF K > 78 THEN
                          GO TO NEXT
                     ELSE
                          GO TO AGAIN;
                     END;
                SPWORD:= " ";
                J:=0;
                L:= K;
                WHILE J <= RE DO
                BEGIN
                     SPWORD(J|1):= INPUT(L|1);
                     J:= J+1;
                     L:= L+1;
                     IF L > 78 THEN GO TO NEXT;
                     END;
                IF SPWORD = RESTART THEN
                BEGIN
                     LINE:= LINE-1;
                     GO TO DESTROY;
                     END;
                IF INPUT(K|1) = "*" THEN SCAN1
                ELSE
                     IF INPUT(K|1) = "#" THEN SCAN1
                     ELSE
                          SCAN2;
                END;
          END;
NEXT:   IF ¬ERROR THEN
     BEGIN
          WRITE(OUTPUT);
          INTFIELDSIZE:=4;
          FOR I:= 1 UNTIL 15 DO WRITEON(BLANK);
          WRITEON(LINE);
          END;
     ERROR:= FALSE;
DESTROY:  END SCAN;
```

```
COMMENT  ************************************************
             MAIN PROGRAM BEGINS HERE.
           ************************************************;
         OPTIONS;
         FOR I:= 0 UNTIL RWN DO
         BEGIN
              READCARD(RW(I)(0|80));
              READCARD(RW(I)(80|80));
              END;
         FOR I:= 0 UNTIL (SC-1) DO
              READ(S(I)(0|3));
         LINE:=0;
         READCARD(INPUT);
         WRITE(BLANK);
         WRITE(BLANK);
         WRITE(INPUT);
         WHILE INPUT(0|3) ¬= "EOP" DO
         BEGIN
              OUTPUT:= " ";
              LINE:= LINE+1;
              IF INPUT = OUTPUT THEN
              BEGIN
                   WRITE(OUTPUT);
                   INTFIELDSIZE:=4;
                   FOR I:= 1 UNTIL 15 DO WRITEON(BLANK);
                   WRITEON(LINE);
                   END
              ELSE
                   SCAN;
              READCARD(INPUT);
              WRITE(BLANK);
              WRITE(BLANK);
              WRITE(BLANK);
         WRITE(INPUT);
              END;
         END;
END.
```

BIBLIOGRAPHY

1. Martin, J., Telecommunications and the Computer, Prentice-Hall, 1969.

2. Davidson, L., A Push Button Telephone for Alphanumeric Input, Datamation, v. 12, pp 27-30, April, 1966.

3. Awad, E. M., Business Data Processing, 3rd ed., Prentice-Hall, 1971.

4. Conway, R. W. and Morgan, H. L., Tele-CUPL: A Telephone Time Sharing System, Communications of the ACM, v. 10, pp 538-542, September, 1967.

5. Abramson, N., Information Theory and Coding, McGraw-Hill, 1963.

6. Auerbach Information, Incorporated, Auerbach Guide to Data Communications, 1970.

7. McKeeman, W. M., Horning, J. J., and Wortman, D. B., A Compiler Generator, Prentice-Hall, 1970.

8. U. S. Naval Postgraduate School Programming Manual for CS 3111, A Formal Definition of Basic, G. A. Kildall, June, 1970.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	Instructor R. C. Bolles Department of Mathematics Naval Postgraduate School Monterey, California 93940	1
4.	LCDR John McKay 1853 47th Street Birmingham, Alabama 35208	1

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author)

Naval Postgraduate School
Monterey, California 93940

2a. REPORT SECURITY CLASSIFICATION

Unclassified

2b. GROUP

3. REPORT TITLE

Tele-Coder
A System for Coding and Decoding Programming Languages for Use With a Push Button Telephone

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

Master's Thesis; September 1971

5. AUTHOR(S) (First name, middle initial, last name)

Lieutenant Commander John Norman McKay, United States Navy

6. REPORT DATE

September 1971

7a. TOTAL NO. OF PAGES

7b. NO. OF REFS

8a. CONTRACT OR GRANT NO.

b. PROJECT NO.

c.

d.

9a. ORIGINATOR'S REPORT NUMBER(S)

9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)

10. DISTRIBUTION STATEMENT

Approved for public release; distribution unlimited.

11. SUPPLEMENTARY NOTES

12. SPONSORING MILITARY ACTIVITY

Naval Postgraduate School
Monterey, California 93940

13. ABSTRACT

The push button telephone has been employed in many systems as a computer terminal with a very restricted capability and in at least one instance as a computer terminal for a general purpose programming language. This thesis discusses the input, output, coding, and decoding problems when considering a general purpose programming language for use with a push button telephone. Included is a general discussion of a system that would use a syntactic analysis and the context of a programming language to produce a code for use with a push button telephone. The output from this analysis would be used to build a table-driven translator to decode the language. An example of such a system is also included.

KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
CODER						
DECODER						
PUSH BUTTON TELEPHONE						
TOUCH TONE TELEPHONE						
TELEPHONE INPUT						
TERMINAL						
TRANSLATOR						
REMOTE ACCESS SYSTEM						

BINDERY
20 JUN 78 24556

Thesis 131305
M2124 McKay
c.1 Tele-coder; a system for coding and decoding programming languages for use with a push button telephone.
BINDERY
BINDERY
BINDERY
20 JUN 78 24556

e-

Thesis 131305
M2124 McKay
c.1 Tele-coder; a system for coding and decoding programming languages for use with a push button telephone.

thesM2124
Tele-coder :
3 2768 001 88205 3
DUDLEY KNOX LIBRARY